

JEE - Java Enterprise Edition

Transactions



Java Enterprise Edition Transactions – an example

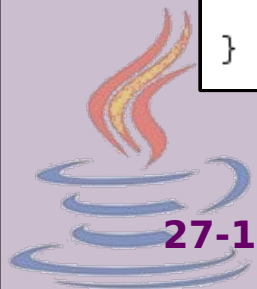
Case Study: Cruise line *Titan Cruises*

```
public TicketDO bookPassage(CreditCardDO card, double price)
    throws IncompleteConversationalState {

    if(customer == null || cruise == null || cabin == null) {
        throw new IncompleteConversationalState();
    }

    try{
        Reservation reservation = new Reservation(
            customer, cruise, cabin, price);
        entityManager.persist(reservation);

        processPaymentEJB.processPaymentByCredit(customer, card, price);
        TicketDO ticket = new TicketDO(customer, cruise, cabin, price);
        return ticket;
    } catch (Exception e) {
        throw new EJBException(e);
    }
}
```



Java Enterprise Edition Transactions

- In **business**, a transaction usually involves an **exchange between two parties**
- E.g. every time a **paying action** is involved, a **transaction** takes place
- **4 characteristics** of a transaction must be met for a **system** to be considered **safe**
- **Transactions** must be **atomic, consistent, isolated** and **durable (acid)**.



Java Enterprise Edition Transactions

- **4 characteristics** for transactions:
 - **Atomic:** An atomic transaction must execute completely or not at all. Every task within a unit of work must execute without errors, otherwise the task is aborted and all changes are made undone.
 - **Consistent:** Refers to the integrity of the underlying datastore. This characteristic must be enforced by the transactional system and the developer. The developer has to ensure, that the database system has appropriate constraints



Java Enterprise Edition Transactions

- **4 characteristics** (continued):
 - **Isolated:** Isolation means that a transaction must be allowed to execute without interference from other processes or transactions. That means, that the data a transactions accesses, cannot be affected by any other part of the system.
 - **Durable:** All data changes made during the course of the transaction must be written to some type of physical storage before the transaction is completed. This ensures that no data gets lost in case of a system crash.



Java Enterprise Edition Transactions

```
public TicketDO bookPassage(CreditCardDO card, double price)
    throws IncompleteConversationalState {

    if(customer == null || cruise == null || cabin == null) {
        throw new IncompleteConversationalState();
    }

    try{
        Reservation reservation = new Reservation(
            customer, cruise, cabin, price);
        entityManager.persist(reservation);

        processPaymentEJB.processPaymentByCredit(customer, card, price);
        TicketDO ticket = new TicketDO(customer, cruise, cabin, price);
        return ticket;
    } catch (Exception e) {
        throw new EJBException(e);
    }
}
```

Java Enterprise Edition Declarative Transaction Mngmt

- With **declarative transaction management**, the transactional behaviour of EJBs can be controlled using the ***@javax.ejb.TransactionAttribute*** annotation
- **Transactional behaviour** of an EJB can be **changed without changing the business logic**

➔ **Reduces complexity** of transactions

➔ Leads to **robust applications**



Java Enterprise Edition Transaction Scope

- The ***transaction scope*** refers to those **EJBs and entity beans**, that are **participating in** a particular **transaction**
- In the example the **scope** of the transaction **starts**, when a **client invokes** the `bookPassage()` **method**
- **Once** the transaction scope is **started**, it is **propagated** to the `EntityManager` and the `ProcessPaymentEJB`
- In a transaction, all tasks making up a unit-of-work must succeed for the transaction to succeed
- In EJB, **tasks** are **expressed as bean-, or service-methods**

Java Enterprise Edition Transaction Scope

- **To trace the scope** of a transaction, simply **follow the thread of execution**
- the **scope includes** every **transaction-aware service** touched during a method invocation
- Not only the thread of execution determines whether an EJB or service is included in the scope of a transaction...
- The **bean's *transaction attributes*** also play a vital role
- A bean's transaction attributes can be accomplished implicitly using annotations



Java Enterprise Edition Transaction Attributes

- The **@javax.ejb.TransactionAttribute** annotation

```
package javax.ejb;  
  
public enum TransactionAttributeType  
{  
    MANDATORY,  
    REQUIRED,  
    REQUIRES_NEW,  
    SUPPORTS,  
    NOT_SUPPORTED,  
    NEVER  
}
```

6 types of
TransactionAttributes

```
package javax.ejb;  
  
import ...  
  
@Target({ElementType.METHOD, ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface TransactionAttribute  
{  
    TransactionAttributeType value() default TransactionAttributeType.REQUIRED;  
}
```

- If not specified the TransactionAttribute is **REQUIRED**

Java Enterprise Edition

Transaction Attributes

- **6 transaction attributes:**
 - **NOT_SUPPORTED:** Transaction is suspended until method is completed
 - **SUPPORTS:** The method is included in the transaction scope, if invoked within a transaction. Method can also be invoked outside of a transaction
 - **REQUIRED:** Method must be invoked within the scope of a transaction. If invoked outside, a new transaction is started covering only this method invocation



Java Enterprise Edition

Transaction Attributes

- **6 transaction attributes** (continued):
 - **REQUIRES_NEW**: A new transaction is always started. If invoked within an existing transaction scope, the transaction is suspended until this method completes
 - **MANDATORY**: The method must be made part of an existing transaction scope, otherwise an ***EJBTransactionRequiredException*** is thrown
 - **NEVER**: The method must not be invoked withing the scope of a transaction, otherwise an ***EJBException*** is thrown



Java Enterprise Edition Transactions

- The **EJB specification** strongly **advises** that **EntityManagers** are only **accessed within the scope of a transaction**. So use one of **REQUIRED**, **REQUIRES_NEW** or **MANDATORY** transaction attributes for methods encapsulation EntityManager access
- **Message-driven beans** may **only** declare one of **NOT_SUPPORTED** and **REQUIRED**, as other attributes don't make sense

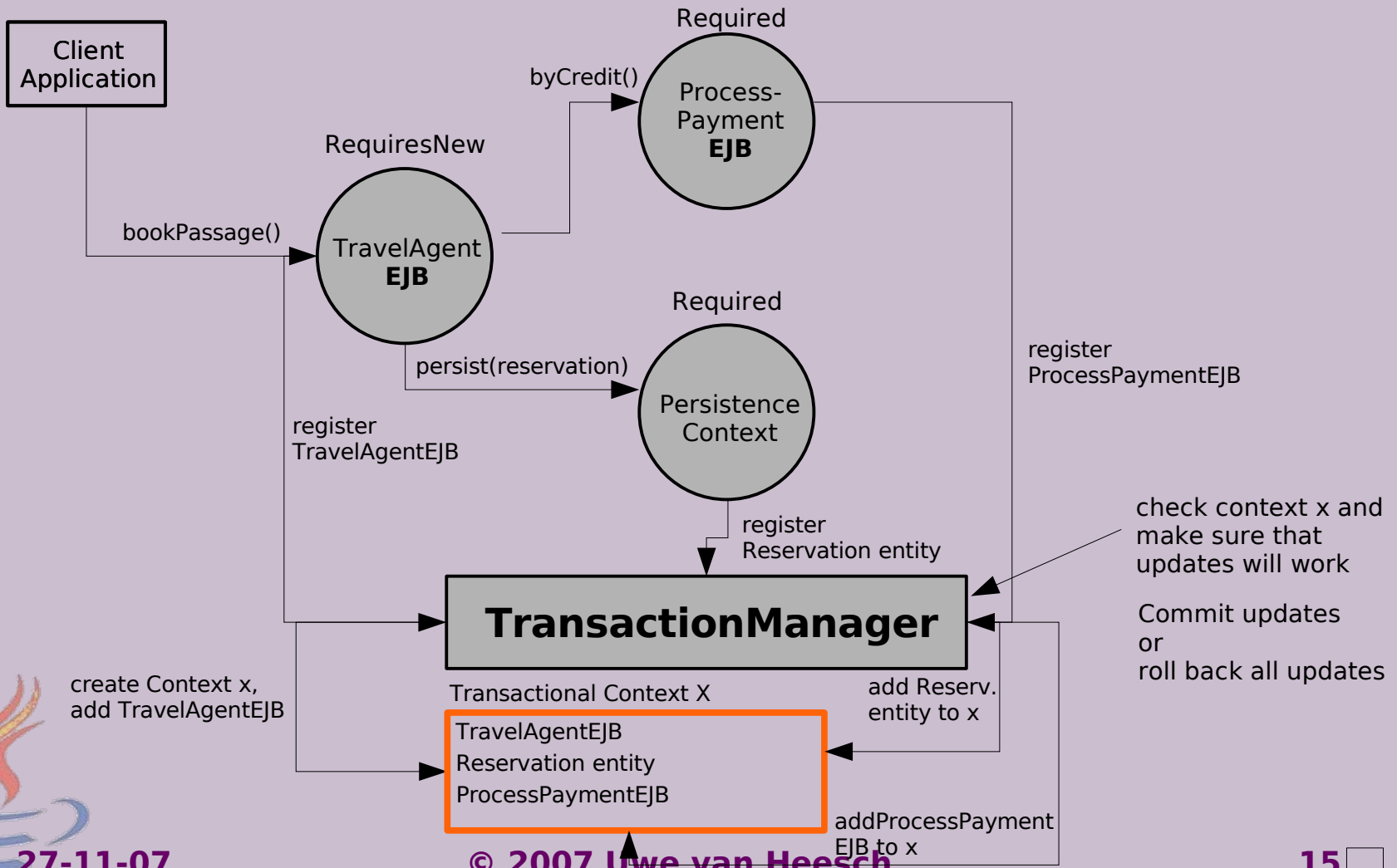


Java Enterprise Edition Transaction Propagation

- As a **transaction monitor**, an **EJB Server** watches each method call in a **transaction**
- **If any updates fail, all** updates to all EJBs and entities will be **reversed**, or ***rolled back***
- **In cases** in which the **container implicitly manages transactions**, commit and rollback decisions are **handled automatically**



Java Enterprise Edition Transaction Propagation



Java Enterprise Edition Transactions & Persistence

- **Some rules** have to be considered when **invoking** in **different EJBs** within the **same transaction** that use **entity managers**
 - invoked **outside a transaction scope**, the **EntityManager creates** a persistence **context** for the duration of the method call. **After** the method completes, all **entities** produced by the call **become detached**
 - if **invoked** from **within a transaction**, the persistence **context is associated** with that transaction



Java Enterprise Edition Transactions

- Beans providing a kind of stateless service, without manipulating data or changing the state of other EJBs do not need to meet the ACID requirements, even if they are called as Utility Beans from within a transaction
- Such beans could be marked with the transaction attribute **NOT_SUPPORTED**
- **Explicit Transaction Management** is possible using the **Java Transaction Service (JTS)**. Then the **programmer himself** must take care of handling the transaction scope by himself, which is quite **error-prone** and thus **not recommendable**

Java Enterprise Edition

Transactions and Exceptions

- **RuntimeExceptions** and subclasses
container handles exceptions **automatically**
 - active transactions are rolled back
 - exception is logged (somehow, unspecified)
 - EJB instance is discarded -> **Garbage Collector**
- **Application Exceptions** (Checked Exceptions)
 - by default, do not cause a transaction to roll back
 - Check Exceptions can be tagged with the **@javax.ejb.ApplicationException** annotation



Java Enterprise Edition Transactions and Exceptions

```
package nl.fontys.jee.interceptors.session;

import javax.ejb.ApplicationException;

@ApplicationException(rollback=true)
public class PaymentException extends java.lang.Exception {

    public PaymentException() {
        super();
    }

    public PaymentException(String msg) {
        super(msg);
    }
}
```



Java Enterprise Edition

In the next lesson

- We will talk about **Isolation and Database Locking**
 - Dirty Reads
 - Repeatable Reads
 - Phantom Reads
- The conditions describe what happens when **two or more transactions operate on the same data**



JEE - Java Enterprise Edition

Transactions

